COMPUTER SYSTEMS LABORATORY

STANFORD UNIVERSITY · STANFORD, CA 94305-2192

Microsupercomputers: Design and Implementation

Stanford University
Computer Systems Laboratory

Technical Progress Report

September 1987 - March 1988

Principal Investigator
John L. Hennessy

Associate Investigator
Mark A. Horowitz

AD-A194 712

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
MICROSUPERCOMPUTERS: DESIGN AND IMPLEMENTATION

5. TYPE OF REPORT & PERIDD COVERED
Semiannual Technical Progress Report
September 1987 - March 1988

6. PERFDRMING ORG. REPORT NUMBER

7. AUTHOR(s)
John L. Hennessy and
Mark A. Horowitz

8. CONTRACT OR GRANT NUMBER(s)
N00014-87-K-0828

9. PERFDRMING DRGANIZATIDN NAME AND ADDRESS
Stanford University
Department of Electrical Engineering
Stanford, CA 94305

10. PRDGRAM ELEMENT, PROJECT, TASK AREA & WDRK UNIT NUMBERS
R&T Project Code: 4331685

11. CDNTROLLING OFFICE NAME AND ADDRESS
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209-2308

12. REPORT DATE
April 1988

13. NO. OF PAGES
26

14. MONITORING AGENCY NAME & ADDRESS (if diff. from Controlling Office)
Office of Naval Research
Computer Science Division, Code: 1133
800 N. Quincy St., Arlington, VA 2217-5000

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATIDN/DOWNGRADING SCHEDULE
None

16. DISTRIBUTIDN STATEMENT (of this report)
Approved for public release. Distribution Unlimited.

17. DISTRIBUTIDN STATEMENT (of the abstract entered in Block 20, if different from report)

18. SUPPLEMENTARY NOTES
Publications noted in bibliography attached.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Computer Architecture, Parallel Programming, CAD Tools.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
Summary of technical progress.

DD FORM 1 JAN 73 1473
EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATIDN OF THIS PAGE (When Data Entered)

Table of Contents

Technical Progress Report

September 1987 - March 1988

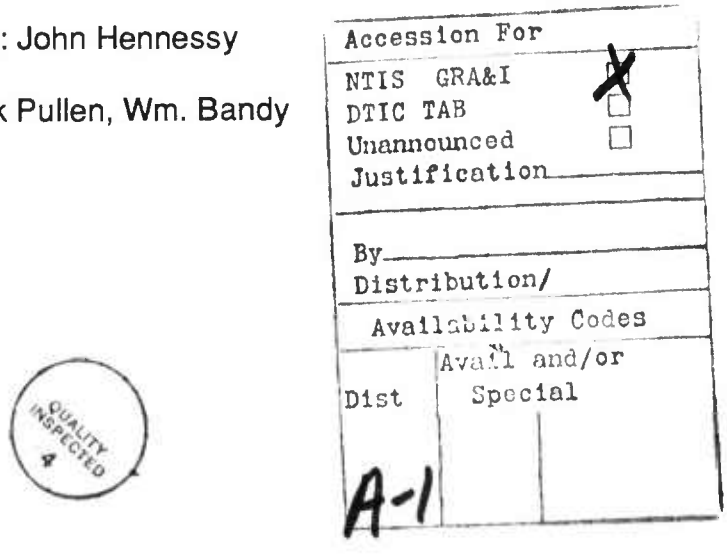
Contract No. N00014-87-K-0828

Order No. 1133

R & T Project Code: 4331685

Principal Investigator: John Hennessy

Monitored by Major Mark Pullen, Wm. Bandy



This work is supported by the Defense Advanced Research Projects Agency and Office of Naval Research.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

Technical Progress

The major progress to be reported for this period is:

1 Parallel Processor Architecture

Work in multiprocessor architecture involves three different aspects. First, we are studying the behavior of parallel applications to understand their behavior and to design architectures to support them. Second, we are exploring a particular set of shared-memory architectures that appear to have substantial scalability. Third, we continue to explore high performance cache design, which plays a vital role in the use of a high-performance processor in a multiprocessor.

1.1 Characteristics of Parallel Programs

This effort attempts to understand how parallel programs share data and how they synchronize. We are measuring not only frequency but also the characteristics of interacting references and synchronization events. We have built two complete measurement systems and are working on a third.

The first system is based on Agarwal's ATUM system [Agarwal 88a], using microcode to trace a multiprocessor VAX. The main limitation is that only applications whose process count is not much greater (say within a factor of 2) of the processor count can be used.

1.2 Multiprocessor Applications

We now have parallel memory reference traces for 3 applications from a 4-processor VAX-8350 computer. Preliminary results of the analysis are reported in the paper "Memory Reference Characteristics of Multiprocessor Applications under MACH" by Anant Agarwal and Anoop Gupta to appear in SIGMETRICS 1988 [Agarwal 88b].

We have also completed a program based on the VAX T-bit that running on a uniprocessor that enables us to get traces corresponding to a multiprocessor with a large number of processors. Traces from this program have also been obtained and are currently being analyzed in greater detail. One of the challenges we now face is to obtain and develop interesting applications that can usefully exploit large numbers of processors.

The two earlier systems suffer because they cannot be extended to large numbers of processes or because doing so would yield a system that runs much too slow. Our new system uses compiled simulation, creating timestamp points at synchronization intervals. The initial version of this system can only collect data on synchronization

operations, but it has already produced significant insights [Davis 88]. The new system will trace both data and synchronization for large applications with hundreds of processes. Accuracy of the trace information is a key goal!

1.3 Scaleable Shared Memory Multiprocessors

One of the major challenges in building a shared-memory multiprocessor with a large number of processors is designing a suitable *cache consistency* mechanism. This is necessary so that each processor can keep local copies of shared data in its cache and yet still see memory data that is consistent with that seen by the other processors. Cache consistency schemes used in existing multiprocessors rely on a bus-based interconnect between the processors to "broadcast" memory addresses corresponding to recently-changed data. Since physical limitations rule out a single bus for interconnecting a large number of processors, we are studying *directory-based* cache consistency schemes as a feasible alternative for a machine with a more general interprocessor network. These techniques maintain a directory associated with main memory that indicates which processors currently contain a cached copy of a given data item.

We are focusing our efforts on several important issues concerning directory-based protocols. First, we are using address traces collected from multiprocessors running parallel applications to simulate the operation of different cache consistency schemes under realistic conditions [Agarwal 88c]. This information allows us to compare and evaluate the network traffic generated by these schemes. Second, we are studying the hardware implementation details of the directory mechanism to determine the design complexity and the area cost incurred with this technique. Finally, we are studying the effect of the directory-based protocols on correct multiprocessor execution in the presence of write buffering, a queueing scheme used in many uniprocessors to improve memory performance.

1.4 High Performance Cache Design

In a high performance multiprocessor, design of the cache is critical to reducing the amount of bus traffic. However, using bus traffic as the only metric leads to designs that may not be the most efficient (since they may raise access time of the cache or the misspenalty). A methodology for examining all factors collectively has been created. Using extensive trace data, some of the conventional wisdom about cache design has been shown to be flawed [Przybylski 88].

Staff: *R. Simoni, J. Hennessy, M. Horowitz, A. Gupta, H. Davis, S. Przybylski, A. Tucker, A. Agarwal*

2 Parallel Software

Our efforts in this arena concentrate in developing innovative parallel applications for use in studying both the properties of parallel applications and for studying "compilation" strategies. The other thrust of our work is on techniques for compiling parallel programs.

2.1 Multiprocessor Applications

On the applications front, the PROTEAN application that we were working on is now running on the 16 processor Encore Multimax. We have been experimenting with granularity and speed-up tradeoffs, and the effects of varying the granularity at run-time. Preliminary results are reported in the paper "Exploiting Variable Grain Parallelism at Run-time" [Gupta 88]. As an extension, we have been trying to port the PROTEAN application to the 64-processor NCUBE at Stanford, but because of both software and hardware problems with the NCUBE, we have not succeeded so far.

2.2 Parallel Programming and Parallel Compilation

This research concentrates on techniques to exploit parallelism. We assume that parallelism will be available at multiple levels, typically at different grain sizes. Furthermore, we have shown that efficient programming demands a subtle tradeoff between exploiting parallelism (particularly at a fine grainsize) and the reduction on running time potentially achieved by exploiting parallelism. Thus, we believe that fine and medium grained parallelism must be controlled in an automated fashion. We are developing techniques for doing this using both conventional languages and using a single-assignment language with large amounts of implicit parallelism. (The latter work is largely supported by the National Science Foundation.)

Our work on single-assignment languages concentrates on Sisal. In earlier work we developed an automatic partitioning system based on machine and application characteristics and capable of dealing with a wide range of shared and nonshared memory machines. Recent work has concentrated on efficient compilation of these languages to make them competitive with conventional languages [Gharachorloo 88], [Gopinath 88] and also on actually implementing the partitioning system for real hardware. This latter effort has produced some impressive early results on speed-up.

Staff: *A. Agarwal, A. Gupta, A. Tucker, J. Hennessy, H. Davis*

3 Computer-Aided Design (CAD) Tools

3.1 Synthesis

For the past several months we have been working on a system for *high level synthesis* of digital hardware called **HERCULES** [De Micheli 88]. We envision synthesis as consisting of two phases: *behavioral synthesis* which involves structurally independent optimizations, and *structural synthesis* which transforms a behavior into a structure which may be implemented. HERCULES transforms the behavioral specification of hardware in the form of C-like programming description through a series of abstractions, with the final result being a logic implementation of the hardware. At each level of abstraction, optimizing transformations are performed which will allow an exploration of the design space available to the designer.

We address the hardware description problem, along with the abstractions and transformations for synthesis. In particular, we developed a method called the *Reference Stack* that may be used during behavioral synthesis to resolve conditional assignment, multiple assignment, variable and constant unfolding, and elimination of cycles for assignments to local variables. We also describe control in terms of a sequencing graph that supports multiple threads of control to be active simultaneously. Several benchmark examples from the High Level Synthesis Workshop held in January 1988 on Orcas Island, Washington is passed through the system: MC6502, Intel8251, and FRISC, a 16-bit RISC type microprocessor. HERCULES is implemented in C on UNIX. There are approximately 23,000 lines of code in the implementation.

3.1.1 Hardware Synthesis

We devised a behavioral modeling language called *ILSP*. The ILSP compiler has been written and is operational. The synthesis system displays the data/control flow graph extracted from a functional model on the window screen. We are currently working on resource optimization.

3.2 Simulation

Simulation now requires vast amounts of cpu time. This severly limits the size of a design that can be tested thoroughly. Incremental and parallel simulation are possible solutions to these current limits. We are investigating the performance of parallel and incremental digital simulation. Specific synthesis and analysis tools are also proposed.

3.2.1 Incremental Simulation

We proposed two incremental simulation algorithms, the *incremental-in-space* and *incremental-in-time* algorithms, and implemented them in our *THOR* simulation system.

The *incremental-in-space* algorithm simulates the circuit components affected by design changes since the previous simulation [Hwang 88]. The *incremental-in-time* [Choi 88] algorithm simulates a circuit component only for the simulation time frames when its inputs make different state transitions from the previous simulation run, maximally utilizing the past history of simulation thus reducing the number of component evaluations to a minimum. Both algorithms are found efficient, showing up to 30x speedups over conventional event-driven simulation. These two algorithms are comparable to each other: one shows better performance for some circuits over the other, depending on the circuit structure and topology of the circuit under simulation.

3.2.2 Parallel Simulation Study

Two parallel algorithms for logic simulation have been developed and implemented on a general purpose shared-memory parallel machine. The first algorithm is a synchronous version of a traditional event-driven algorithm which achieves speed-ups of 4 to 6 with 8 processors. The second algorithm that has been developed is new and totally asynchronous [Soule 88]. There are no synchronization locks or barriers between processors and the problems of massive state storage and deadlock have been eliminated. This allows the processors to work independently at their own speed on different elements and at different times.

Staff: *G. De Micheli, D. Ku, S.Y. Hwang, T. Blank*

3.3 Power and Gnd Noise

We have developed a system called Ariel for analyzing voltage drops and current density in the power networks of CMOS VLSI circuits. Three main parts, a Magic-based resistance extractor, a Rsim-based current simulator, and a network analyzer, work in tandem to examine the current/voltage characteristics of the power networks with a minimum of manual effort from the designer [Stark 88].

Ariel gets its resistor networks from Magic's resistance extractor, which uses a simple version of polygonal reduction. The extractor calculates resistance by counting the number of squares between two connection points in a region and multiplying by the sheet resistance. To find the currents that flow in the power supplies, Ariel uses RSIM to find which nodes change, the time-constant for each change, and which transistor is driving the node. The information on the current is then feed into a third program that calculates the voltages on power supplies. This program first breaks the loops in the power nets to make them into trees. Although this only approximates the actual voltage drop, it is guaranteed to be conservative and drastically reduces the time needed to determine the voltage on the power buses.

Staff: *M. Horowitz, D. Stark*

3.4 Placement and Routing with Parallel Processing

The Locus Project is concerned with achieving better quality automatic layout of integrated circuits by making use of the increased computational power of multiprocessors. This work is primarily supported by a Center for Integrated Systems seed grant, and partially by DARPA. The basic idea is to get better placement quality by using the actual routing as the measure of goodness for each potential placement. Routing itself is a computationally intensive task; hence the need for multiprocessors.

The first step of the Locus Project is nearing completion: the program LocusRoute, a parallel global router for standard cells has been developed [Rose 88a], [Rose 88b], [Rose 88c]. It achieves parallelism along three orthogonal "axes" of parallelism: routing several wires at once, routing segments of a wire in parallel, and dividing up the potential routes of a segment among different processors to be evaluated. The implementation of two of these approaches achieve significant speedup: wire-by-wire parallelism attains speedups from 6.9 to 13.6 using sixteen processors, and route-by-route achieves up to 4.6 using eight processors. When combined, these approaches can potentially provide speedups of as much as 55 times.

Some work related to placement optimization has also been done. We have investigated the equilibrium dynamics of Simulated Annealing-based placement optimization, and developed a reliable method for "measuring" the "temperature" [Rose 88d] of a placement. This method can be used to determine the starting temperature in placement systems that switch from a non-annealing based strategy to an annealing-based one.

Staff: *J. Rose, J. Hennessy*

4 VLSI

Our work on VLSI has been focused on new circuit structures that might be useful for future high-speed processors / floating-point units. In this effort we are exploring BiCMOS as well as MOS designs.

4.1 RAM Design

We have been experimenting with both very high speed and high density RAM design. In high speed RAMs we have designed, fabricated, and tested a 3.5ns 4K bit ECL I/O BiCMOS sRAM. [Yang 88] The design uses a novel memory cell that contains a bipolar transistor along with a CMOS latch. The bipolar transistor's collector is connected to the nwell so it occupies very little space; the cell is only 30% larger than a standard 6T cell. Using only small swing bipolar logic in the access path provides the fast access time. The cell also has independent read and write ports allowing simultaneous read and write operations. The device was fabricated at IDT on a 1.5u BiCMOS technology.

In high density RAMs we have designed, fabricated and tested a 16K dRAM using a 1T RAM cell. The memory uses a relatively large cell (.1pf, and 19x14 lambda) but is still about 3-4 times denser than a 6T design. The cell should be scalable down to a 1.2u technology without major change. Scaling from 2u to 1.6u leaves the cell capacitance roughly unchanged because of the large change in gate oxide thickness (40nm -> 25nm) The memory has an access time of 22ns (RT) and a cycle time a little over 30ns in a 2u technology. A conservative design approach was used, there are no bootstrapped nodes, and the timing chain is self-timed whenever possible. The net result is an access time that is slower than it could be, but is quite robust. We have tested the noise margins by adjusting the voltage on the dummy cell and have found that the memory has large noise margins. We are now planning to write a module generator for dRAMs so they can be used on other chips.

Staff: *M. Horowitz, R. Kao, T.S. Yang*

4.2 BiCMOS

We are ramping up our effort to design circuits in high-performance BiCMOS and bipolar technologies. We have already designed a 3.5ns BiCMOS sRAM and have a few adder circuits in fabrication. To drive the design process we have started looking at the design of a high-performance floating point unit. Our goal is to build a co-processor capable of running at 100MHz and starting a new add or multiply every cycle. With this goal in mind we are beginning to look at building the major blocks that this chip requires: multiplier, adder, shifter, control logic, registerfile, bus drivers, latches and control logic.

To support this design effort we have begun working on a set of CAD tools to support bipolar and BiCMOS design. We were able to modify the Magic layout system to support bipolar devices, and now have a number of different BiCMOS technology files.

Unfortunately almost all of the simulation tools can't deal with the resulting simulation file, so the BiCMOS RAM was the first chip in a number of years that was not "switch" simulated before fabrication. We had simulated the pieces of the design using SPICE, but were not able to simulate the entire chip. To fill this hole we are working on a BiCMOS simulator, and have a prototype running. The simulator can now handle bipolar circuits and we plan to extend it to handle BiCMOS circuits later this year.

Staff: *M. Horowitz, R. Kao, R. Alverson, D. Stark, D. Wingard*

4.3 Multiplication

The demand for high performance floating point coprocessors has created a need for high-speed, small-area multipliers. Array multipliers achieve the highest performance but have a large silicon cost, while shift and add multipliers require very little hardware but have lower performance. We have used an iterative partial tree structure to provide a high-performance, small-area multiplier [Santoro 88]. The clock is generated internally and is set to match the delay through two carry-save adders and a latch. To complete a 64x64 multiply requires only seven clocks, and a new multiply can be started after 4 clocks. The 1.6u parts clock at over 80MHz.

Staff: *M. Horowitz, M. Santoro*

4.4 Single-Chip Testers

We have completed the testing of our high performance CMOS pin electronics. This circuit has been fabricated in a 2u technology and is able to generate outputs to drive the DUT with about .5ns resolution. The chip can generate all the needed formats (NRZ RZ RO RT RC) and can drive the output to either of two high and two low levels. For measurement of the DUT outputs, the chip contains an analog comparator that is sampled at the user specified time [Gasbarro 88].

We will use this pin electronics design in the design of a single chip tester that will be sent out for fabrication this year. Each chip can drive 16 DUT pins at over 30MVecs/s with edge resolution on each pin of about .5ns. The chip contains a dRAM for vector storage (40Kbits) and a decompressor that allows the chip to effectively store about 10K vectors/pin. The chip has a simple asynchronous interface that allows it to be attached to a number of different buses, and only requires 70 pins. Using these chips, it should be possible to build an IMS class tester (or better) using only 8 chips. Also since the test electronics are so small they can be located very close to the DUT thus eliminating the problem with cables and reflections.

Staff: *M. Horowitz, J. Gasbarro*

References

[Agarwal 88a] Agarwal, A., Sites, R.
Multiprocessor Address Tracing and Characterization Using ATUM.
In *15th International Symposium on Computer Architecture*.
IEEE/ACM, Honolulu, HI, June, 1988.
To appear.

[Agarwal 88b] Agarwal, A., Gupta, A.
Memory-Reference Characteristics of Multiprocessor Applications under MACH.
In *SIGMETRICS*. IEEE, 1988.
To appear.

[Agarwal 88c] Agarwal, A., Simoni, R., Hennessy, J., Horowitz, M.
Scaleable Directory Schemes for Cache Consistency.
In *15th International Symposium on Computer Architecture*. IEEE, Honolulu, HI, June, 1988.
To appear.

[Choi 88] Choi, K., Hwang, S.Y., Blank, T.
Incremental-in-Time Algorithm for Digital Simulation.
In *25th Design Automation Conference*. IEEE/ACM, Anaheim, CA, June, 1988.
To appear.

[Davis 88] Davis, H., Hennessy, J.
Characterizing the Synchronization Behavior of Parallel Programs.
In *Sym. on Parallel Programming: Experience with Applications, Languages and Systems*. ACM , New Haven, CT, July, 1988.
To appear.

[De Micheli 88] De Micheli, G., Ku, D.
HERCULES - A System for High-Level Synthesis.
In *Design Automation Conference*. IEEE/ACM, Anaheim, CA, June, 1988.
To appear.

[Gasbarro 88] Gasbarro, J., Horowitz, M.
Integrated Pin Electronics for VLSI Functional Testers.
In *Custom Integrated Circuits Conference*. IEEE, Rochester, NY, May, 1988.
To appear.

[Gharachorloo 88] Gharachorloo, K., Sarkar, V., Hennessy, J.
A Simple and Efficient Implementation Approach for Single Assignment Languages.
In *Lisp and Functinal Programming Conference*. ACM, Salt Lake City, UT, July, 1988.
To appear.

[Gopinath 88] Gopinath, K.
Copy Elimination with Abstract Interpretation.
Computer Science Department Classic 87-17, Stanford University, February, 1988.

[Gupta 88] Gupta, A., Tucker, A.
Exploiting Variable Grain Parallelism at Runtime.
In *Sym. on Parallel Programming: Experience with Applications, Languages, and Systems.* ACM, New Haven, CT, July, 1988.
To appear.

[Hwang 88] Hwang, S.Y., Blank, T., Choi, K.
Fast Functional Simulation: An Incremental Approach.
IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems , July, 1988.
To be published.

[Przybylski 88] Przybylski, S., Horowitz, M., Hennessy, J.
Performance Effects in Memory Hierarchy Design.
In *15th International Symposium on Computer Architecture.* IEEE, Honolulu, HI, June, 1988.
To appear.

[Rose 88a] Rose, J.S.
The Parallel Decomposition and Implementation of an Integrated Circuit Global Router.
In *Sigplan Symposium on Parallel Programming.* ACM, New Haven, CT, July, 1988.
To appear.

[Rose 88b] Rose, J.S.
LocusRoute: A Parallel Global Router for Standard Cells.
In *25th Design Automation Conference.* IEEE/ACM, Anaheim, CA, June, 1988.
To appear.

[Rose 88c] Rose, J.S.
LocusRoute: A Parallel Global Router for Standard Cells.
In *Workshop on Placement and Routing.* MCNC, Atlanta, GA, May, 1988.
To appear.

[Rose 88d] Rose, J.S., Klebsch, W., Wolf, J.
Equilibrium Detection and Temperature Measurement of Simulated Annealing Placements
In *Workshop on Placement and Routing.* MCNC, Atlanta, GA, May, 1988.
To appear.

[Santoro 88] Santoro, M., Horowitz, M.
A Pipelined 64x64b Iterative Array Multiplier.
In *International Solid-State Circuits Conference*. IEEE, San Francisco, CA, February, 1988.

[Soule 88] Soule, L., Blank, T.
Parallel Logic Simulation on General Purpose Machines.
In *25th Design Automation Conference*. IEEE/ACM, Anaheim, CA, June, 1988.
To appear.

[Stark 88] Stark, D., Horowitz, M.
Analyzing CMOS Power Supply Networks using Ariel.
In *25th Design Automation Conference*. IEEE/ACM, Anaheim, CA, June, 1988.
To appear.

[Yang 88] Yang, T., Horowitz, M., Wooley, B.
A 4ns 4kxl Two-Port BiCMOS SRAM.
In *Custom Integrated Circuits Conference*. IEEE, Rochester, NY, May, 1988.
To appear.

A 4 nsec 4K×1bit Two-Port BiCMOS SRAM

T.S. Yang, M.A. Horowitz, and B.A. Wooley

Center for Integrated Systems, Stanford University
Stanford, California 94305

ABSTRACT

This paper introduces a two-port BiCMOS static memory cell that combines ECL level word-line voltage swings and emitter-follower bit line coupling with a static CMOS latch to achieve access times comparable to those of high-speed bipolar SRAM's, while preserving the high density and low power of CMOS memory arrays. The memory can be accessed for read and write independently and simultaneously, making it especially attractive for the design of video, cache and other application-specific memories. An experimental 4K×1bit two-port memory integrated in a 1.5μm-5GHz BiCMOS technology exhibits a read access time of 4 nsec and a power dissipation of 550 mW.

INTRODUCTION

The highest speed static memories have generally been realized using advanced bipolar technologies. However, the large cell area and standby power dissipation have precluded the scaling of these circuits to increasingly higher levels of integration. Recently, BiCMOS technology has been used to significantly enhance the speed of CMOS static memory arrays. Most BiCMOS SRAM's described to date combine conventional CMOS cells with the use of bipolar transistors in the sense amplifiers and for driving large capacitive loads[1,2]. In these designs, the access time remains limited by factors such as the large voltage swing on the word lines, the limited cell output current and the number of circuit stages used[3].

In addition to access time itself, simultaneous multiport access capability is becoming an increasingly important feature of high-speed static memories. However, conventional multiport designs typically have both a large cell area and relatively slow access.

This paper introduces a BiCMOS two-port static memory cell, and associated access circuitry, with which it is possible to achieve access times comparable to those of high-speed bipolar SRAM's while retaining the high density and low power of CMOS memory arrays. The cell, referred to as a CMOS Storage Emitter Access (CSEA) cell, combines ECL level word-line voltage swings and emitter-follower bit line coupling with a static CMOS latch. Compared with conventional multiport memory designs, the CSEA memory offers extremely high speed and small size together with the independent read and write access capability, making it especially attractive for multiport memory designs such as video, cache and other application-specific memories. To demonstrate the operation of the cell, a complete 4K×1bit two-port SRAM has been designed and integrated in a 1.5μm BiCMOS technology. The memory operates from a single 5.2V supply with ECL-compatible I/O. A typical read access time of 4 nsec at a power dissipation of 550 mW was achieved in the initial prototypes.

* This research was supported in part by a fellowship from IBM and by DARPA under Contract No. MDA903-83-C0335

BiCMOS CSEA MEMORY CELL

The schematic of the CSEA memory cell and its associated bit line sensing circuit is shown in Figure 1. In the CSEA memory, a small word-line swing of only 550 mV and emitter-follower coupling to the bit line are adopted to minimize the word-line delay and increase the bit-line charging current. A static CMOS latch is retained to reduce the standby power consumption of the array. In Figure 1, the READ word line (RWL) serves as the positive supply for the cell's internal CMOS latch, and the cell is read by raising this word line. When a high output is stored in the cell, the increase in RWL is coupled directly through M2 to the base of the output emitter-follower, Q6, which forms a differential pair with the sense amplifier input transistor, Q7. For a low output stored in the cell, Q6 is turned off by M4 and the bit line current, I_{BL}, is switched to flow through Q7. The cell is written through the pass transistor M5, which is controlled by the WRITE word line. Full CMOS logic levels are used on both the WRITE bit line (WBL) and the WRITE word line (WWL). As a consequence of the single-ended write, care must be taken to avoid disturbing the unselected cells in the row where the WRITE word line is selected. Such write disturbances are avoided by biasing the unselected WRITE bit lines at a level close to the logic threshold of the M2-M4 latch inverter when the READ word line is at its low level.

The area of the CSEA cell is approximately 35% larger than that of a 6-transistor CMOS cell implemented using the same layout rules and design style. The increased area is due primarily to the independent read and write ports, which preclude the sharing of signal lines among adjacent rows and columns. However, the CSEA cell is comparable to other single-ended memory cells[4] and is much smaller than conventional multiport cells using differential bit lines.

MEMORY ARCHITECTURE

The block diagram of a 4Kbit two-port CSEA memory is shown in Figure 2. The memory is organized as a 64-row by 64-columns array. The array is controlled by 64 WRITE row decoders, 64 WRITE column switches, 64 READ row decoders and 32 READ column switches. Each READ column switch selects two READ bit lines and the data are multiplexed at the output buffer by the least significant bit of the READ address. Two sets of address inputs, one for read and the other for write, are available externally thereby allowing direct access to the read and write ports of the CSEA cell array. The separate data input and output paths offer improved system performance by eliminating the need for multiplexing the data bus, thus reducing the I/O delay in the critical path.

Since the loading a cell imposes on its READ word line depends both on the data stored in the cell and on whether or not the cell is being written, the READ word and bit lines are laid out orthogonal to the WRITE bit and word lines, respectively. Therefore, RWL parallels WBL, and the selected word line is loaded by only a single cell with an active WRITE word line. This arrangement serves to minimize write condition interference with the read operation. Finally, in order to isolate power supply coupling and reduce switching noise, the read path, comprised mainly of small-swing ECL circuits, and the write path, consisting of CMOS logic, are powered through separate pads.

CIRCUIT DESIGN

A schematic of the READ path is given in Figure 3. Selection of the READ word line (RWL) is accomplished entirely with current-switching ECL circuits operating at low voltage swings. The logic levels in these circuits are established by means of an on-chip, supply-compensated bandgap reference. Push-pull address input buffers are used to provide fast transitions at the input to the row decoders, and the decoding is accomplished using diode decoders. The word lines are driven by Darlington emitter-followers tied through resistors to a common pull-down current. The active pull-down current available to discharge a deselected word line is 7mA, while a static pull-down current of 125μA is maintained in each of the unselected word lines. To ensure that the bit line reference tracks the high RWL level in the selected cell, the level of the selected word line is monitored with a wired-OR of emitter-followers driven by each of the word lines.

The schematic of the WRITE path is shown in Figure 4 and is similar to that of conventional BiCMOS SRAM's[2]. WRITE address decoding is accomplished by means of dynamic series decoders that are clocked by the write enable signal. The unselected WRITE bit lines are biased at the voltage level of the internal latch threshold, V_{wblf}; this reference is generated from the common pull-down current source in the read path.

EXPERIMENTAL PERFORMANCE

A die photo of the complete 4K×1bit two-port memory is shown in Figure 5; the die size is 2.5mm×3.5mm. The performance of the prototype memory is summarized in Table I. Shown in Figure 6 is an oscillograph of a typical read access at room temperature with a power dissipation of 550 mW. The access time is 4 nsec. At 100°C case temperature the measured worst case read access time is 6 nsec and the power dissipation is 750 mW.

The independent read/write capability is illustrated in Figure 7, where the memory is read and written simultaneously. Additional measurements have shown little data dependency in the read access time and no interference between read and write operations. The setup and hold times of the write address signals with respect to the write enable pulses are less than 1 nsec, and the minimum write enable pulse width is 4 nsec; thus a write cycle time of less than 6 nsec can be achieved. This suggests that the memory can be cycled at a rate close to 200 MHz. Since most of the power in the memory is dissipated in the peripheral circuits, a 16Kbit two-port SRAM implemented using the same circuit techniques and technology is projected to have a typical read access time of 4.5 nsec with a power dissipation of 750 mW.

ACKNOWLEDGEMENTS

The authors wish to thank Integrated Device Technology, Inc. for fabricating the circuits. They are especially grateful to F. C. Hsu and C. C. Wu of IDT for numerous helpful discussions. They are also indebted to J. B. Kuo and R. W. Dutton of Stanford, who provided the initial inspiration for this project.

REFERENCES

[1] Jun-Ichi Miyamoto, et. al., "A High-Speed 64K CMOS RAM with Bipolar Sense Amplifier", *IEEE J. of Solid-State Circuits*,Vol. SC-19; No. 5, p. 557-563; Oct., 1986.

[2] Katsumi Oguie, et. al., "64-kbit ECL RAM Using HI-BICMOS Technology", *IEEE J. of Solid-State Circuits*,Vol. SC-21; No. 5, p. 681-985; Oct., 1986.

[3] Takakuni Douseki and Yasuo Ohmori, "BiCMOS Circuit Technology for A High Speed SRAM", *Symp. on VLSI Circuits Digest of Technical Papers*, p. 77-78; May, 1987.

[4] Kevin J. O'Connor, "The Twin-Port Memory Cell", *IEEE J. of Solid-State Circuits*, Vol. SC-22; No. 5, p.712-720; Oct., 1987.

Technology	1.5μm, 5GHz BiCMOS
Cell Size	26μm x 25μm
Chip Size	2.5mm x 3.5mm
Configuration	4K×1Bit 2-port
I/O Interface	ECL 10K compatible
Power Supply	-5.2 Volts
Power Consumption	550 mW
Read Access Time	4.0 ns
Minimum Write Pulse	4.0 ns

Table I: Performance summary of the 4Kbit CSEA Memory

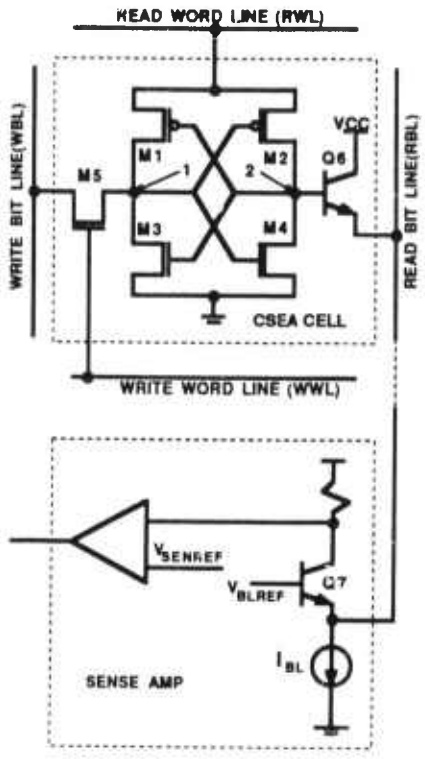


Figure 1: CSEA cell and associated bit line sensing circuit.

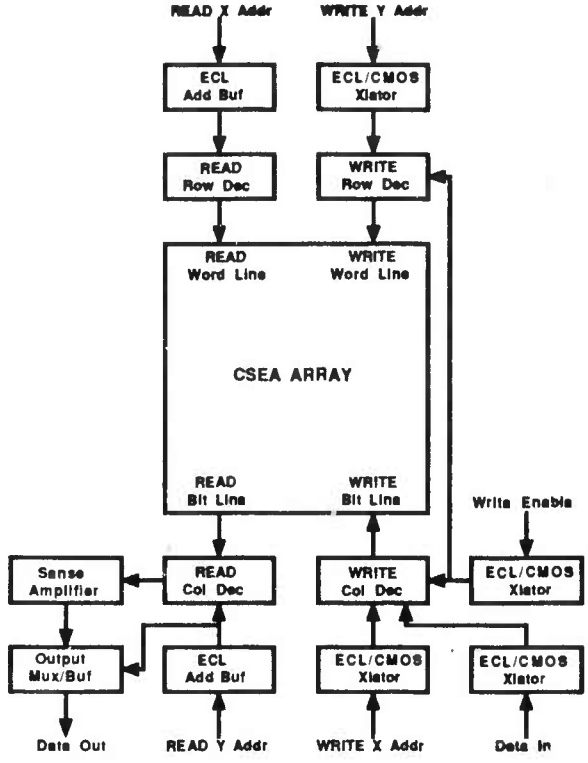


Figure 2: Block diagram for 4Kbit two-port CSEA memory.

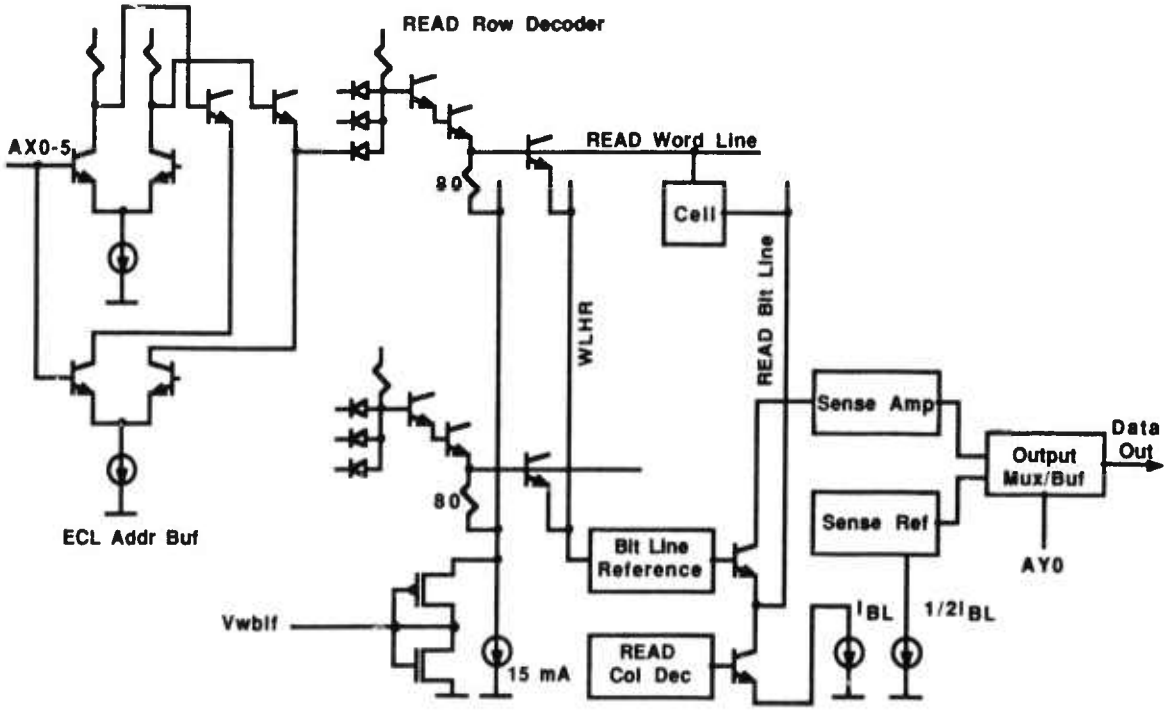


Figure 3: Circuit schematic of the READ path.

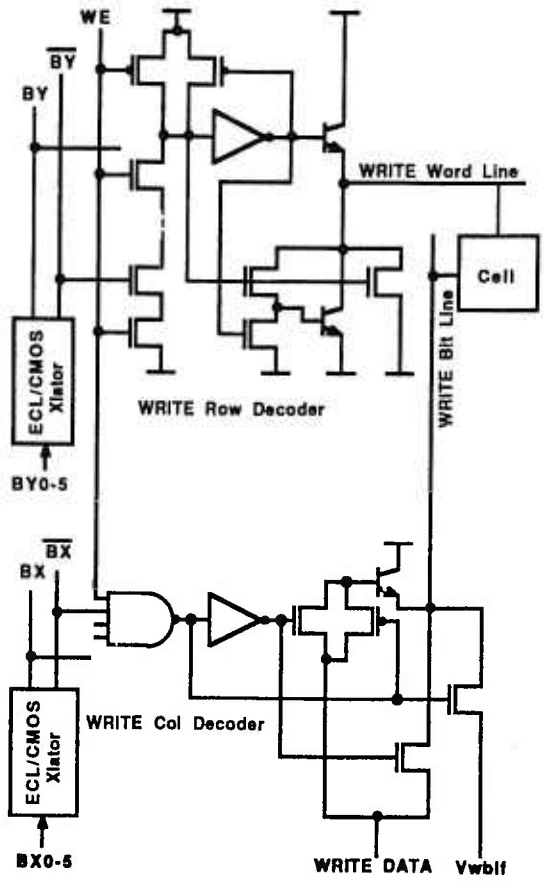


Figure 4: Circuit schematic of the WRITE path.

Figure 5: Die photo of the 4Kbit CSEA memory.

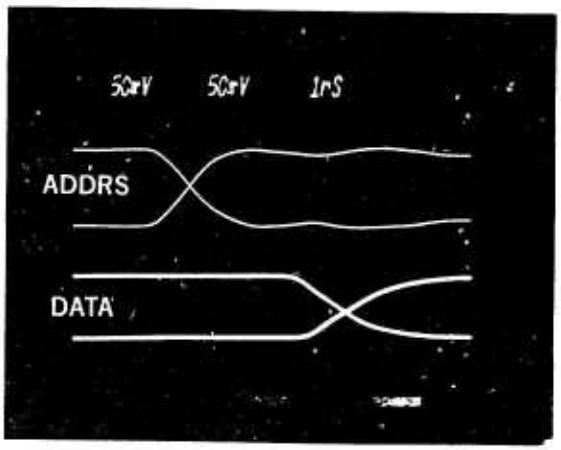


Figure 6: Oscillograph of measured READ access time..

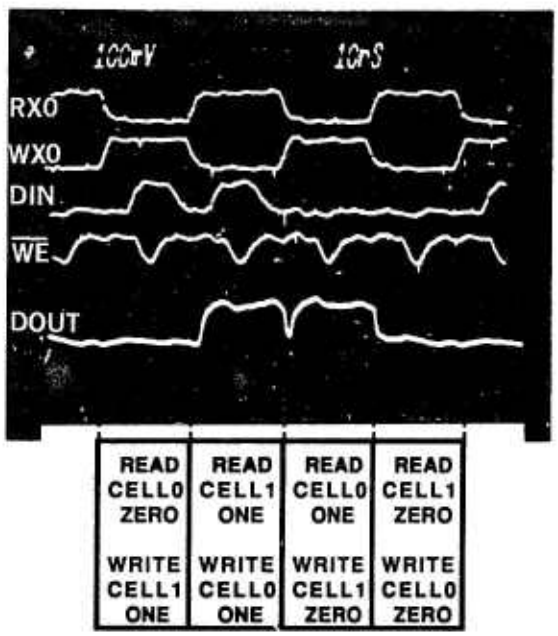


Figure 7: Oscillograph showing independent READ and WRITE memory accesses.

LocusRoute: A Parallel Global Router for Standard Cells

Jonathan Rose
Computer Systems Laboratory
Center for Integrated Systems
Stanford University, Stanford CA 94305

Abstract

A fast and easily parallelizable global routing algorithm for standard cells and its parallel implementation is presented. LocusRoute is meant to be used as the cost function for a placement algorithm and so this context constrains the structure of the global routing algorithm and its parallel implementation. The router is based on enumerating a subset of all two-bend routes between two points, and results in 16% to 37% fewer total number of tracks than the TimberWolf global router for standard cells [Sech85]. It is comparable in quality to a maze router and an industrial router, but is factor of 10 times or more faster. Three approaches to parallelizing the router are implemented: wire-by-wire parallelism, segment-by-segment and route-by-route. Two of these approaches achieve significant speedup - route-by-route achieves up to 4.6 using eight processors, and wire-by-wire achieves from 5.8 to 7.6 on eight processors.

1 Introduction

The best way to evaluate a given placement of circuit modules is to route it and determine the final area. Since routing is a time-consuming task typical placement algorithms [Hana72,Breu77] use other metrics such as total wire length or crossing counts that are easier to calculate. The advent of usable commercial multiprocessors is leading us to consider using more compute-intensive cost functions *if* efficient parallel algorithms can be developed. The aim of the *Locus Project* is to integrate placement and routing into one optimization process, and to do this by using multiprocessing to increase the speed of the routing.

This paper presents the first step in the Locus Project: *LocusRoute*, a new global routing algorithm for standard cells, and its parallel implementation. Our goal is to make the average routing time for one net close to the time that it takes to recalculate more conventional cost functions such as that used in the TimberWolf [Sech85] Simulated Annealing algorithm. The intention is for the global router to be invoked to rip-up and re-route wires whose end points have changed when one or more cells are moved in an iterative improvement placement scheme. This means that routing time must be about one to five milliseconds per net on a VAX 11/780-class machine.

The routing performance of LocusRoute, as measured by total number of routing tracks, is better than that of TimberWolf [Sech85] and comparable to a maze router and an industrial router. It is fast because it investigates only a subset of two-bend routes between pairs of pins to be routed. The routing speed is increased further by parallelizing the algorithm in three ways: routing several wires at once, routing several two-point segments simultaneously, and evaluating possible two-bend routes in parallel. The wire-by-wire parallel approach achieves speedups ranging from 5.8 to 7.6 using 8 processors. The route-by-route approach achieves speedups of up to 4.6 using 8 processors. Since these two "axes" of parallelism are *orthogonal* to each other, their respective speedups will multiply.

This paper is organized as follows: Section 2 reviews related work. Section 3 defines the problem of global routing and gives our routing model. Section 4 describes the LocusRoute algorithm and compares it to other routers. Section 5 presents three approaches for speeding up the new router using parallel processing, and performance results.

2 Related Work

Previous work on parallel routing [Breu81, Blan81, Rute84, and many others] has generally focused on a fixed hardware mapping for the Lee routing algorithm [Lee61]. As such they lack the flexibility that is required in practical CAD software such as the global router described in [Kamb85]. Another drawback of special hardware for the Lee algorithm is that a uniprocessor implementation can be made very efficient using special software data structures that cannot be put easily into fixed hardware. A more flexible approach is to use general purpose parallel processors, which can be adapted to many applications. Using the flexibility of a general purpose multiprocessor, several "axes" of parallelism can be exploited. If these axes are *orthogonal* to each other then when used together they can provide significant speedup. Two approaches to parallelizing an algorithm are said to be orthogonal if, when used together, the resulting speedup is the product of the speedup of the individual methods.

3 Problem Definition and Routing Model

Global routing for standard cells first decides for each group of electrically equivalent pins (pin clusters) which of those pins are actually to be connected. Second, if there is no path between channels when one is required, it must decide either which built-in feedthrough to use or where to insert a feedthrough cell. Lastly, it must determine the channel to use in routing from a pad into the core cells.

In this discussion of global routing there will be no differentiation between feedthrough cells and built-in feedthroughs - they are referred to jointly as *vertical hops*. The decision to insert a feedthrough cell or use a built-in feedthrough is deferred to a post-processing step. This does result in some inaccuracy in the track count. However, using this approximation (and the routing algorithm to be described) the 904-wire Primary1 circuit from the standard cell benchmark suite [Prea87] global routed to 249 tracks, using 995 vertical hops. The actual, post-process track count using 10 feedthrough cells and 985 built-ins was 253, only 1.6% more tracks. For the 3029-wire Primary2 circuit with 3424 vertical hops (287 feedthroughs, 3137 built-ins) the approximate track count was 546 and the post-process count was 590, an increase of 8%.

The usual objective of a global router is to minimize the sum of the channel densities of all the channels (hereafter called the *total density*). It is important to note that the total density can be traded off with the number of vertical hops, so to compare the total density of two global routings fairly they should both use the same number of vertical hops.

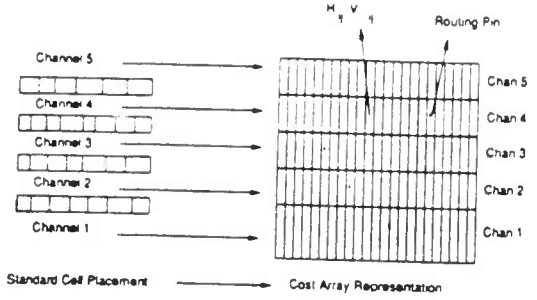


Figure 1 - *Routing Model*

3.1 Routing Model

All of the routing algorithms discussed here are based on the same routing model: Each possible routing position in a channel (also called *routing grid* of that channel) is represented as one element of an array as shown in Figure 1. The array, called the *Cost Array*, has a vertical dimension of the number of rows plus one, and a horizontal dimension of the width of the placement in routing grids. Each element of the Cost Array contains two values: H_{ij} and V_{ij}. H_{ij} contains the number of of wire routes that pass horizontally through the grid at channel i in position j. V_{ij} is the cost, assigned by parameter, of traversing a row in travelling from channel i to channel $i + 1$ at grid position j. A wire is represented as a list of (i, j) pairs of locations in the Cost Array, corresponding to the locations of pins to be joined.

The objective is to find a minimum-cost path for each wire. The wire's cost is given by the sum of all of the H_{ij} and V_{ij} that it traverses. After a path is found for a wire that goes through location (i, j) its presence is recorded in the Cost Array (the appropriate H_{ij} and V_{ij} are incremented) so that subsequent wires can take it into account. The more wires going through a particular location in a channel, the less likely it is that area will be used. Note that in this model the total density is not directly minimized, but rather a combination of average density and wire length.

4 The LocusRoute Algorithm

In this section the uniprocessor LocusRoute algorithm is described, and a performance comparison with other routers is given. There are five steps in the LocusRoute global routing algorithm:

1. A multi-point wire is decomposed into two-point *segments*, using Kruskal's algorithm [Krus56]. This algorithm has running time $O(n^2)$ in the number of pin clusters. The effect of the sub-optimality of this decomposition is discussed in section 4.4 below.

2. The segments are further decomposed, if necessary, into *permutations*, which are the set of possible routes between each pin in a pin cluster.

3. A low-cost path in the Cost Array is found for each permutation by evaluating a subset of the two-bend routes between each pin pair. The permutation with the best cost is selected as the route for that segment.

4. Traceback. This step joins all the segments back together, and assigns unique numbers to distinct segments of the same wire in each channel. This is so that a channel router can distinguish between two segments and will not inadvertently join them together.

5. Wire *lay down*. The presence of the newly routed wire is put into the Cost Array by incrementing the array elements where the new wire resides. Once there, other wires can take it into account.

The details of the second and third steps are described in the following sections. The first and last two are simple enough that the above description suffices.

4.1 Decomposition into Permutations

Each two-point segment consists of pairs of *pin clusters* that contain electrically equivalent pins. The LocusRoute algorithm considers routes between every pin in one cluster and every pin in the other cluster. Each such route is called a *permutation*. Figure 2 illustrates three of the four possible permutations between clusters A and B, which have two pins each. The four possible permutations are: (A_1, B_1), (A_1, B_2), (A_2, B_1), (A_2, B_2). If clusters A and B are separated by only a short horizontal distance, then the (A_1, B_2) permutation is most likely the least-cost path of the four. If the horizontal distance is large then it is possible that any one of the four permutations could have the low-cost path, and hence all should be investigated. This has been confirmed experimentally, and a constant horizontal separation (300 routing grids) has been determined beyond which total density will improve if all four permutations are evaluated.

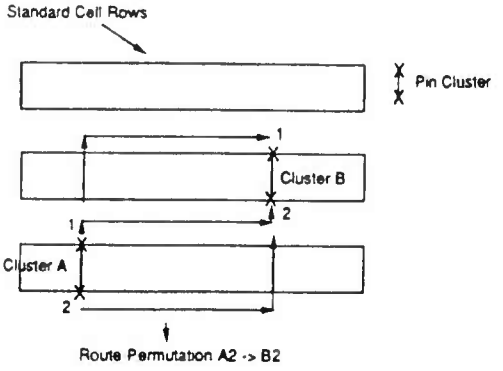


Figure 2 - *Permutation Decomposition of Segment*

4.2 Route Enumeration

The LocusRoute algorithm searches for a low-cost path for a permutation by *enumerating* a number of different routes. The idea is to evaluate the cost of a subset of all two-bend routes between the two pins, and then choose the one with the lowest cost. Generation of two-bend routes is discussed in [Ng86]. Figure 3 illustrates three possible two-bend (or less) routes inside a representation of the Cost Array as a small example.

If the horizontal distance between the two pins is H routing grids, and the vertical difference is C channels then the total number of two-bend routes is $C+H$. A parameter, called the *two bend percent* (TBP) dictates the percentage of the total number possible two-bend routes to be evaluated. Thus the total number of routes evaluated is given by $\frac{TBP}{100} \times (C + H)$.

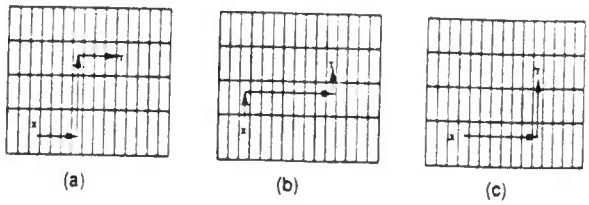


Figure 3 - *Sample Two-Bend Routes*

The priority order of the routes evaluated (when TBP is less than 100) is as follows: first all principally horizontal routes (those with bends only at the left and right extremes) are evaluated. Then the principally vertical routes (those with bends at the upper and lower extremes) are evaluated. Horizontal routes are evaluated first because it is important that all of the potential channels for the route be examined at least once. Within the horizontal and vertical groups, routes are searched in bisection order; i.e. if the limits of the group span are normalize to [0,1] then the routes are prioritized as $0, 1, \frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}$, and so on.

The two-bend evaluation approach was calibrated against a least-cost path maze router between the two points. Note that both routers are not allowed to go beyond the bounding box of the two end points of the segment. This is different than comparing against a maze router for multipoint wires since that is a less constrained problem and the maze router will have more success, as discussed in Section 4.4. Experimentally, it was determined that a TBP of 20% would result in a path as good as that found by the maze router, as compared on the basis of total density for the entire circuit. On all of the test circuits used in the experiments discussed in the section 4.4, the LocusRoute router's total density was within 2% of that obtained by the two-point maze router, with one exception of 3.3%. Most of the differences were below 1%. This is surprising in that the maze router looks for not only two-bend routes but for three or more bend routes. It implies that two-bend routes provide a sufficiently rich route set for the standard cell routing problem.

4.3 Iteration

The LocusRoute algorithm makes use of a general iterative technique in the manner described in [Nair87]. Briefly, this means that after the first time all wires are routed, each is sequentially ripped up from the Cost Array and then re-routed. By routing each wire several times (typically four is sufficient), the wire order-dependency is reduced and the final answer is improved by five to ten percent. Also - of benefit to the end-purpose of integrated placement and routing - the nature of iteration is similar to the placement environment in which wires are ripped up and re-routed many times.

4.4 Uniprocessor Performance Results

This section compares the quality and execution time of LocusRoute with other routers.

Table 1 shows a comparison between the LocusRoute global router and the TimberWolf [Sech85] global router for several industrial circuits. These circuits are from several sources: The standard cell Benchmark suite (Primary1, Primary2, Test06 [Prea87]), Bell-Northern Research Ltd. (BNRA->BNRE), and the University of Toronto Microelectronic Development Centre (MDC). The placement for all of the circuits was done by the ALTOR standard cell placement program [Rose85, Rose88]. The TimberWolf version used was TimberWolf 4.1, obtained in July 1987. LocusRoute shows significantly better total density than does the TimberWolf global router, ranging from 16% to 37% fewer tracks. The principal reason is that the TimberWolf global router is constrained to use only the minimum number of vertical hops, whereas LocusRoute uses considerably more. This is a reasonable practice in current technology because many standard cells contain "free" built-in feedthroughs. The execution times of LocusRoute and TimberWolf are comparable for most of the examples, though TimberWolf is faster by a factor of 8 and 3 respectively for circuits Test06 and Primary2. This is due to the fact that the LocusRoute algorithm increases in running time proportional to the area covered by the wire, which is much larger in these two circuits.

Circuit Name	# Wires	Total Density		
		Locus	TWolf	% Few
BNRE	420	138	179	**22%**
MDC	575	150	179	**16%**
BNRD	774	188	225	**16%**
Primary1	904	262	316	**17%**
BNRC	937	202	247	**18%**
BNRB	1364	320	442	**27%**
BNRA	1634	315	432	**27%**
Test06	1673	335	537	**37%**
Primary2	3029	563	702	**20%**

Table 1 - *Comparison of LocusRoute and TimberWolf*

For comparison purposes a maze router [Lee61] was developed that exhaustively determines the optimal solution to the two-point routing problem as defined in Section 3. Note that it uses the same cost function as the LocusRoute router. It also determines a good approximation to the minimum-cost Steiner tree for multi-point wires using the approach described in [Aker72]. The maze router was carefully optimized for speed. Table 2 shows the comparison of total density and execution time for the maze router and the LocusRoute router, for all of the test circuits. The comparison is made on the basis of roughly equal numbers of vertical hops. Execution times are for four iterations over all wires on a DEC Micro Vax II.

Circuit Name	Total Density			Time (Micro Vax IIs)		
	Locus	Maze	Diff	Locus	Maze	Factor
BNRE	138	129	7%	88	2378	27x
MDC	150	141	6%	178	3173	18x
BNRD	188	182	3%	167	3306	20x
Primary1	262	255	3%	325	6534	20x
BNRC	202	189	7%	363	7250	20x
BNRB	320	308	4%	599	15116	25x
BNRA	315	294	7%	769	19652	26x
Test06	335	316	6%	5137	92272	18x
Primary2	563	549	3%	3758	48295	13x

Table 2 - *Comparison of LocusRoute and Maze Router*

For all circuits the LocusRoute total density (total number of routing tracks) is no greater than 7% more than that achieved by the maze router, and in some cases is as little as 3%. Most of this difference is due to the sub-optimality of dividing the wires up into two point nets. LocusRoute is markedly faster than the maze router - ranging from 13 to 27 times faster. This gain in speed is more than worth the increase in total density for the end-purpose of integrated placement and routing.

For two of our circuits, we can also compare the total routing density with the United Technologies global router used in the recent benchmark effort at the 1987 Physical Design Workshop [Prea87,Robe87]. The placements used above for circuits Primary1 and Primary2 were also routed by the UT router. Table 3 shows the comparison of total density for both circuits, with each router using roughly the same number of vertical hops. The total density of the UT router for circuit Primary1 is notably less than for the LocusRoute router. This is probably due to the fact that the UT router also performs neighbour exchanges and cell orientation changes on the placement in order to reduce the total number of tracks. The LocusRoute total density for circuit Primary2 is slightly less than that achieved by the UT router. We have no information on the execution time of the UT router, except that for circuits near the size of Primary2, it would take roughly 10000 Vax 11/780 seconds [Robe87].

Circuit Name	# Wires	Total Density	
		LocusRoute	Benchmark
Primary1	904	253	194
Primary2	3029	560	562

Table 3 - *Comparison of LocusRoute and Benchmark Router*

5 Parallelization

In this section several ways of parallelizing the LocusRoute router are proposed and implemented:

1. Wire-based Parallelism. Each processor is given an entire multi-point wire to route.

2. Segment-based Parallelism. Each two-point segment produced by the Kruskal decomposition can be routed in parallel.

3. Permutation-based Parallelism. Each of the four possible permutations, as discussed in Section 4.1, can be evaluated in parallel.

4. Route-based Parallelism. Each of the possible two-bend routes for every permutation can be evaluated in parallel.

Note that these are only *potential* axes of parallelism. It is possible to eliminate some of them as uneconomical by using statistical run-time measurements of the sequential router. For example, the number of two-point segments that actually need to have all four permutations evaluated is quite small with respect to the total. Thus, permutation-based parallelism is not going to provide significant speedup and isn't worth the time it requires to develop. Other measurements, however, show that the time spent evaluating the cost of two-bend routes ranges from 50 to 90 percent of the total routing time, so that some amount of speedup from route-based parallelism can be expected.

The following sections gives the details of three axes of parallelism, their performance and a quantitative measure of the of degradation in quality if there is some.

5.1 Wire-Based Parallelism

In Wire-Based parallelism, each multi-point wire is given to a separate processor, which runs the LocusRoute routing algorithm as described in Section 4. Thus, each processor executes the following "flow" for a different wire: Prior to decomposition, if the iteration technique is used, the wire must be "ripped up" out of the Cost Array. Next, each wire is decomposed into two-point nets, and possibly further into permutations. A subset of the potential two-bend routes is generated, and then evaluated by traversing the Cost Array. When a final route is chosen, the Cost Array is updated to reflect the new presence of that route.

The Cost Array is a shared data structure to which all processors have read and write access. This is an excellent axis of parallelism: if the sharing of the Cost Array does not cause performance degradation due to memory contention, the speedup should simply be the number of wires that are routed in parallel. The resulting parallel answer, however, will not necessarily be the same as the sequential answer. The problem is that the sequential router has complete knowledge of all wires that have already been routed, by virtue of their presence in the cost array. The parallel router has less information because it doesn't see the wires that are being routed simultaneously. The more wires routed in parallel, the less information each processor has to choose good routes that avoid congestion and hence the total density increases. The total density will increase as the number of processors increases. The measured effect on total density is discussed below, in Section 5.1.1.

An interesting issue is whether or not each processor should lock the Cost Array as it both rips up and re-routes wires in the Cost Array. The act of ripping up a route is essentially a decrement, and re-routing is an increment on a cell in the Cost Array. Locking the Cost Array during these operations (to ensure that two simultaneous operations on the same element does not prevent one of the operations from being lost) can cause a serious performance degradation. However, the final routing quality did not decrease when locking was omitted. The reason for this is that the probability of two processors accessing the same Cost Array element (of which there are many) at the same instant is very low. Even if very few increment or decrement operations are lost, the effect on final quality is negligible since only a few elements would be wrong by a small amount.

5.1.1 Wire-Based Parallel Results

Figure 4 is a plot of the speedup versus number of processors for the 904-wire (Primary1) example running on an eight-processor Encore MULTIMAX. The speedup for p processors, S_p is calculated as $\frac{T_1}{T_p}$, where T_1 is the execution time on one processor and T_p is the execution time using p processors. The Encore uses National 32032 chip sets which, in our benchmarks, timed out slightly faster than a DEC Micro Vax II.

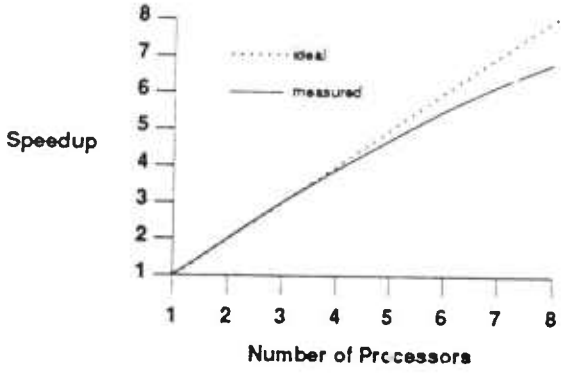


Figure 4 - *Wire-Based Speedup for Circuit Primary1*

Note that the execution time is only the actual routing computation time, excluding input time. The "knee" in the curve at five processors occurs because on an eight-processor Encore two processors share one cache. When five or more processors are used, pairs of processors interfere with each other more. For this circuit the increase in total density (between 1 and 8 processors) is negligible, and the number of vertical hops increases about 3%.

Table 4 gives the speedup using eight processors for the other test circuits. The speedup ranges from 5.8 for a smaller circuit to 7.6 for the largest. The speedup is less for smaller circuits because they are done in such a short time, and the startup overhead becomes a factor. The execution time is for four iterations over all the wires. It was discovered that very large global wires, such as TRUE or FALSE that have up to 150 pins, caused a severe degradation in speedup. This is because our system handles those nets just like any other, and the $O(n^2)$ nature of the Kruskal algorithm causes load balancing problems. Since most production systems treat TRUE and FALSE signal nets differently (usually tapping directly into the power lines with special cells) these were eliminated under the assumption that they could be handled quickly that way.

Table 5 gives the density and vertical hop counts for both 1 and 8 processors using wire-based parallelism. The degradation in total density ranges between .7% to 7.6%. The increase in vertical hops is generally 3% or less, with one exception. In the placement context this level of degradation is tolerable, though we have considered two ways of reducing the problem. The first is to try to ensure that the different processors only deal with wires that are in distinct physical areas, so that the wires routed simultaneously do not interact. This approach was not implemented because in the placement context (with incremental placement "moves") the wires are most likely to be in the same area and can't be separated.

Circuit Name	1-Proc Time (s)	8-Proc Time (s)	8-Proc Speedup
BNRE	78	13	5.8
MDC	88	15	5.9
BNRD	156	22	7.0
Primary1	321	47	6.8
BNRC	221	33	6.7
BNRB	697	92	7.6
BNRA	878	124	7.1
Test06	6261	869	7.2
Primary2	4334	574	7.6

Table 4 - *Wire-Based Parallelism Speedup*

The second way to reduce processor interference is not to rip up a route until the new route is determined. In this way there is a much shorter period of time in which the cost array does not contain the presence of the wire. Unfortunately, this severely degrades the new route of the wire itself since it sees the old copy of itself when new routes are being evaluated. Experimentally, the degradation was found to be bad enough to nullify any gain from the approach.

5.2 Segment-Based Parallelism

In segment-based parallelism, each two-point segment of a wire is given to a different processor to route. This is the stage following the Kruskal decomposition, but prior to the evaluation of different two-bend routes. Measurements of the sequential router showed that about 60% of the routing time was spent on wires with more than one segment. This means that a speedup of about two might be expected using three processors. Even though there are many wires that provide two or three-way parallel tasks, however, the size of those tasks are not necessarily equal. The amount of time taken by the LocusRoute router to route two points is proportional to the Manhattan distance between the two points. If, in a three-point wire, two of the points are close together and the third is far away, it will then take much longer to route one segment than the other. The processor assigned to the short segment will be idle while the longer one is being routed. This unequal load prevents a reasonable speedup. On the test circuits a speedup of about 1.1 using two processors was measured.

Circuit Name	Density		Vertical Hops	
	1-Proc	8-Proc	1-Proc	8-Proc
BNRE	129	135	454	470
MDC	134	144	243	243
BNRD	181	185	528	562
Primary1	262	264	934	958
BNRC	193	199	739	749
BNRB	312	326	1897	1953
BNRA	300	311	2103	2154
Test06	325	336	3196	3253
Primary2	560	584	3022	3097

Table 5 - *Wire-Based Parallelism Quality*

It is fairly clear, however, that an extra processor could be assigned to a *number* of processors that are routing different wires. It is likely that at any given time, one of them will be able to use the extra processor to route an extra segment. This technique would become essential in wire-based parallelism if the number of processors were on the order of the number of wires. In that case, the load balance would become a problem because wires with many segments take much longer than wires with few segments. Hence segment-based parallelism could be used as a method to balance those loads.

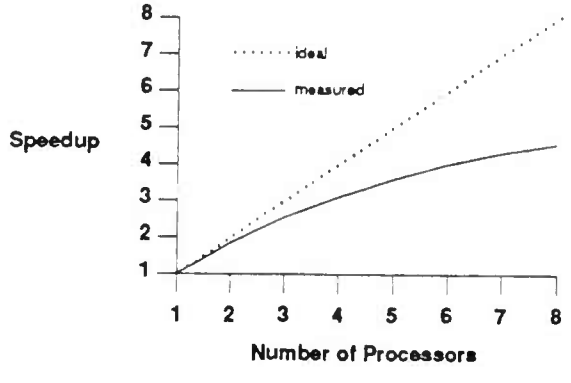


Figure 5 - *Route-Based Speedup for Test06*

5.3 Route-Based Parallelism

In route-based parallelism all of the two-bend routes to be evaluated are divided among processors. Each finds the lowest-cost path among the set of two-bend routes that it is assigned. When all processors finish, the route with the best overall cost is selected. In this case the processor loads are well balanced because the routes are all of the same length, and the number of routes is evenly divided among the processors.

Figure 5 is a plot of the speedup versus number of processors for the circuit Test06, a large circuit. It achieves a speedup of 4.6 using 8 processors.

Table 6 gives the best speedup achieved for all of the test circuits, ranging from 1.2 using 2 processors to 4.6 using 8 processors. The principal reason for the limitation in speedup is the sequential portion of the routing: the wire decomposition and the post-route processing that places the presence of the route into the Cost Array. On the small circuits that have lesser speedup, the sequential portion is about 50% of the total routing time, while on the larger circuits which have better speedup the sequential portion ranges from 10-15%. Another reason is that some segments have only one potential route, limiting parallelism.

Circuit Name	Best Route-based Speedup (Speedup/#Processors)
BNRE	1.2/2
MDC	1.3/2
BNRD	1.4/2
Primary1	1.8/3
BNRC	1.6/3
BNRB	2.1/4
BNRA	2.0/4
Test06	3.6/5, 4.6/8
Primary2	3.3/5

Table 6 - *Performance of Route-Based Parallelism*

5.4 Combining Two Axes of Parallelism

The wire-based parallel and route-based parallel approaches are perfectly orthogonal; hence their speedups should "multiply". Assume, for a given circuit that a speedup of S_w is achieved using wire-based parallelism on W processors, and a speedup of S_r is achieved using route-based parallelism on R processors. Then, because the two approaches are orthogonal, the resulting speedup when they are used together should be $S_w \times S_r$ using $W \times R$ processors. This model neglects the effect of memory contention that may occur when the number of processors is increased dramatically. Table 7 shows the best predicted speedup for the test circuits. Combined speedup ranges from 7 using 16 processors to 33 using 64 processors. The smaller circuits are routed very quickly and so it is difficult to get speedups greater than 10 due to the startup overhead. The larger circuits benefit greatly from the combination of the approaches.

Table 7 also contains the average routing time per net on one processor, A_1, and what the the average routing time per net would be under the maximum speedup, A_{RW}. That is, $A_{RW} = \frac{A_1}{S_w \times S_r}$. The average routing times for all circuits, under the various speedups range from 5.0ms to 28ms, and approaches our goal of one to five milliseconds per net.

Circuit	$\frac{S_w}{W}$	$\frac{S_r}{R}$	$\frac{S_w \times S_r}{W \times R}$	A_1 (ms)	A_{RW} (ms)
BNRE	$\frac{5.8}{8}$	$\frac{1.2}{2}$	$\frac{7.0}{16}$	46	6.6
MDC	$\frac{5.9}{8}$	$\frac{1.3}{2}$	$\frac{7.7}{16}$	38	5.0
BNRD	$\frac{7.0}{8}$	$\frac{1.4}{2}$	$\frac{9.8}{16}$	50	5.1
Primary1	$\frac{6.8}{8}$	$\frac{1.8}{3}$	$\frac{12.2}{24}$	89	7.2
BNRC	$\frac{6.7}{8}$	$\frac{1.6}{3}$	$\frac{10.7}{24}$	59	5.5
BNRB	$\frac{7.6}{8}$	$\frac{2.1}{4}$	$\frac{16.0}{32}$	127	8.0
BNRA	$\frac{7.1}{8}$	$\frac{2.0}{4}$	$\frac{14.2}{32}$	134	9.5
Test06	$\frac{7.2}{8}$	$\frac{4.6}{8}$	$\frac{33}{64}$	935	28
Primary2	$\frac{7.6}{8}$	$\frac{3.3}{5}$	$\frac{25}{40}$	358	14

Table 7 - *Predicted Combined Speedup of Wire and Route Parallelism*

5.5 Conclusions

A new global routing algorithm for standard cells and its parallel implementation has been presented. The LocusRoute algorithm users significantly fewer tracks than the TimberWolf standard cell global router, and is comparable to a maze router and an industrial router. It is more than a factor of 10 faster than either of the two latter routers. Three axes of orthogonal parallelism were developed to speed up the LocusRoute router further. Two of the three axes that were implemented achieved significant speedup - up to 7.6 using eight processors and 4.6 using eight processors. They should produce combined speedups of up to 33 times.

In the future, the combined approach will be run on a multiprocessor with more processors. Using a sophisticated scheduling algorithm we hope to do better than simple multiplication of speedups. The Locus placement environment is currently being developed, and will be combined with the LocusRoute global router. Our aim is to achieve smaller final area by using the global routing as a better measure of each placement.

Acknowledgements

The author is grateful to Tom Blank who provided many good suggestions for this paper. Thanks also to Grant Martin of Bell-Northern Research for the use of company circuits and to the people involved in the standard cell benchmark effort for supplying those test circuits. Carl Sechen provided version 4.1 of TimberWolfSC.

6 References

[Aker72]
S. B. Akers, "Routing," Chapter 6 of Design Automation of Digital Systems; Theory and Techniques, M.A. Breuer, Ed., Englewood Cliffs, NJ, Prentice-Hall, 1972.

[Blan81]
T. Blank, M. Stefik, W. VanCleemput, "A Parallel Bit Map Processor Architecture FOR DA ALGORITHMS," Proc. 18th Design Automation Conference, June 1981, pp. 837-845.

[Breu77]
M.A. Breuer, "Min-Cut Placement," Journal of Design Automation and Fault-Tolerant Computing, Oct 1977, pp 343-362.

[Breu81]
M.A Breuer, K. Shamsa, "A Hardware Router," Journal of Digital Systems, Vol IV, Issue 4, 1981, pp. 393-408.

[Hana72]
M. Hanan, J.M. Kurtzberg, "Placement Techniques," Chapter 4 of Design Automation of Digital Systems; Theory and Techniques, M.A. Breuer, Ed., NJ, Prenuce-Hall, 1972.

[Kamb85]
T. Kambe, T. Okada, T. Chiba, I. Nishioka, "A Global Routing Scheme for Polycell LSI," Proc. ISCAS 1985, pp. 187-190.

[Krus56]
J.B. Kruskal, "On The Shortest Spanning Subtree of a graph and the Traveling Salesman Problem," Proc. Amer. Math. Soc, 7, 1956, pp. 48-50.

[Lee61]
C.Y. Lee, "An Algorithm for Path Connections and Its Applications." IRE Transactions on Electronic Computers, Vol EC-10, pp 346-365, 1961.

[Nair87]
R. Nair,"A Simple Yet Effective Technique for Global Wiring," IEEE Transactions on Computer-Aided Design, Vol CAD-6, No. 2, March 1987, pp. 165-172.

[Ng86]
A P-C Ng, P. Raghavan, C.D. Thompson, "A Language for Describing Rectilinear Steiner Tree Configurations," Proc. 23rd Design Automation Conference, June 1986, pp. 659-662.

[Prea87]
B.T. Preas, "Benchmarks for Cell-Based Layout Systems," Proc. 24rd Design Automation Conference, June 1987, pp. 319-320.

[Robe87]
Ken Roberts used the United Technologies Standard Cell global router on the standard cell benchmark placements. Results were discussed at the 1987 DAC.

[Rose85]
J.S. Rose, W.M. Snelgrove, Z.G. Vranesic, "ALTOR: An Automatic Standard Cell Layout Program," Proc. Canadian Conference on VLSI, November 1985, pp. 168-173.

[Rose88]
J.S. Rose, W.M. Snelgrove, Z.G. Vranesic, "Parallel Standard Cell Placement Algorithms with Quality Equivalent to Simulated Annealing," IEEE Trans. on CAD, Vol. CAD-7, No. 3, March 1988, pp. 387-396.

[Rute84]
R.A. Rutenbar, T.N. Mudge, D.E. Atkins, "A Class of Cellular Architectures to Support Physical Design Automation," IEEE Trans. on CAD, Vol. CAD-3, No. 4, October 1984, pp. 264-278.

[Sech85]
C. Sechen, A. Sangiovanni-Vincentelli, "The Timberwolf Placement and Routing Package," IEEE JSSC, Vol. SC-20, No. 2, April 1985, pp 510-522. pp. 432-439.